Abstract Model-Based Checkpoint and Recovery

Gergely Pintér

The most frequent cause of service unavailability in computer systems is related to transient hardware or software faults. A common way for addressing these issues is based on introducing a checkpoint and recovery schema. Checkpoint generation means the periodic saving the process state into a stable storage. This image can be used for restarting the process from the previously saved state reducing this way the processing time loss to the interval between the checkpoint creation and the failure. Although the core idea is relatively simple, its implementation can get very complicated in case of complex internal data structures since the representation of an object model in the non-volatile storage and the transmission between the memory and the storage requires significant programming effort.

This paper aims at proposing an automatic code generation scheme that provides a transparent and platformindependent facility for persisting object structures of arbitrary complexity in stable storage.

There are several function libraries supporting the checkpoint creation and state recovery directly. These approaches typically do not address the problem of storage and reconstruction of complex data structures, usually provide the capability of saving and loading unstructured blocks of memory only [1]. These low-level methods make the persistent storage of many interconnected objects difficult and error-prone. The solution proposed in [2] depends on a service of the UNIX operating system for saving the entire memory image and register set in a file. The drawback of this strategy is the dependence on a specific operating system feature therefore lack of portability.

The object serialization facility provided by popular languages and class libraries should be taken into consideration as well. Classes implementing the (empty) Serializable interface in the Java language are persisted transparently by the framework. This powerful and elegant feature is not portable to other languages. The Microsoft Foundation Classes library provides a C++ base class with a virtual member function Serialize that should be overridden by subclasses. Although the core idea is portable, this approach is no more than a coding convention that enables the seamless integration to the framework requiring the programmer to explicitly implement the serialization routines.

To put it together the approaches discussed here can not deal with complex data structures, rely on non-portable features or require significant programming effort therefore are not feasible for our purposes. A sophisticated checkpoint and recovery system should be based on the identification of core data modeling concepts instead of exploiting nonportable platform-specific features, a transparent mapping to the non-volatile medium and on a generic object-oriented pattern for implementation. The proposal should enable the automatically generation of low-level data exchange routines.

In our approach the data modeling concepts of the Meta Object Facility (MOF) were used that identifies four fundamental artifacts: classes representing key modeling concepts with appropriate attributes, aggregation and reference relations and inheritance. Classes can be collected into packages.

The mapping to the non-volatile medium is similar to the one specified by the XML Metadata Interchange (XMI) standard. Objects are mapped to XML sub-trees. The name of the class and the package hierarchy containing it is coded in the name of the XML node. The node is labeled with an XML attribute specifying the unique textual identifier of the instance. The class attributes are sub-nodes named after the name of the attribute. Sub-trees according to aggregated objects are recursively embedded in the nodes of the appropriate container instance. References are represented by empty XML nodes containing the textual identifier of the referenced instance.

Our design pattern consists of three key classes. An abstract base class provides some basic

housekeeping functionalities for application-specific classes. A singleton model class acts as the container of objects that are not explicitly contained by other instances. Checkpoint and recovery means this way the storage and retrieval of the single model instance. Maintaining unique textual identifiers and matching them to objects is also the responsibility of the model class. The serialization of application-specific classes is performed by the factory class. The straightforward mapping to XML according to the XMI conventions enables the automatic generation of the serialization routines.

References

[1] Y. Huang and C. Kintala, "Software fault-tolerance in the application layer," in Software Fault Tolerance, M. R. Lyu, Ed., pp. 231-248. John Wiley and sons, 1995.

[2] J. S. Plank, M. Beck, G. Kingsley, and K. Li, "Libckpt: Transparent Checkpointing under UNIX," Tech. Rep. UT-CS-94-242, 1994.